# MOHID-LAGRANGIAN: Tracers and kernel adding guide

## 1. Summary Overview

This short guide will guide you through the creation of new tracers or particle types. If you are familiarized with the concept of Lagrangian tracer or Dynamical systems theory each particle is represented by a “state vector” where each vector component is a variable describing a characteristic of the particle. A characteristic could be the position, the mass, the size, or any other property that evolves on time.

Each physical process affecting these physical properties, changes those vector component values according to a function describing the physics on that process. The functions will be called *kernels*. The concept of *kernel is* a borrowed conceptfrom the GPU. In GPU programming a kernel is an independent function that perform operations over an entire array. Here, we approach this concept in such a way that the kernels will be functions that operates over an entire array of tracers particles at once.

Here, we will briefly explain which changes should be done in the code in order to introduce a new particle with a new physics.

In summary, there are two main classes that will address the creation of a new particle:

* Tracer: It contains the class describing the physical properties of the particle.
* Kernel: It contains the functions describing the physical processes that affect the tracer properties.

At this moment, the MOHID-Lagrangian can simulate Lagrangian particles, paper, and plastic particle types in the ocean.

## 2. Create a new particle type

To guide through the creation of a new particle type, we will do it through an example. We will create a “coliform” class, a bacteria tracer type. Despite having the common physical properties that any other particle has (such as the position, for example), we will include other properties and processes to describe their growing in size and resuspension from the bottom.

### 2.1 Creating new files.

So, we will need to create two main files.

1. **tracerColiform.f90**
2. **kernelColiform.f90**

The tracerColiform.f90 can be copy from tracerPlastic.f90 or even from tracerUser.f90. The tracerUser.f90 is an empty template to fill with the different tracer properties.

Once **we copied the tracerPlastic.f90 file and renamed to tracerColiform.f90, we must edit the particle properties (and replace Plastic references with Coliforms of course) to fit them with our desired properties**. In this example we will use some invented properties to illustrate the example.

We will modify the plastic particle properties definition (if you have some doubts review the file tracerPlastic.f90) to consider those properties important for the coliform. For example:

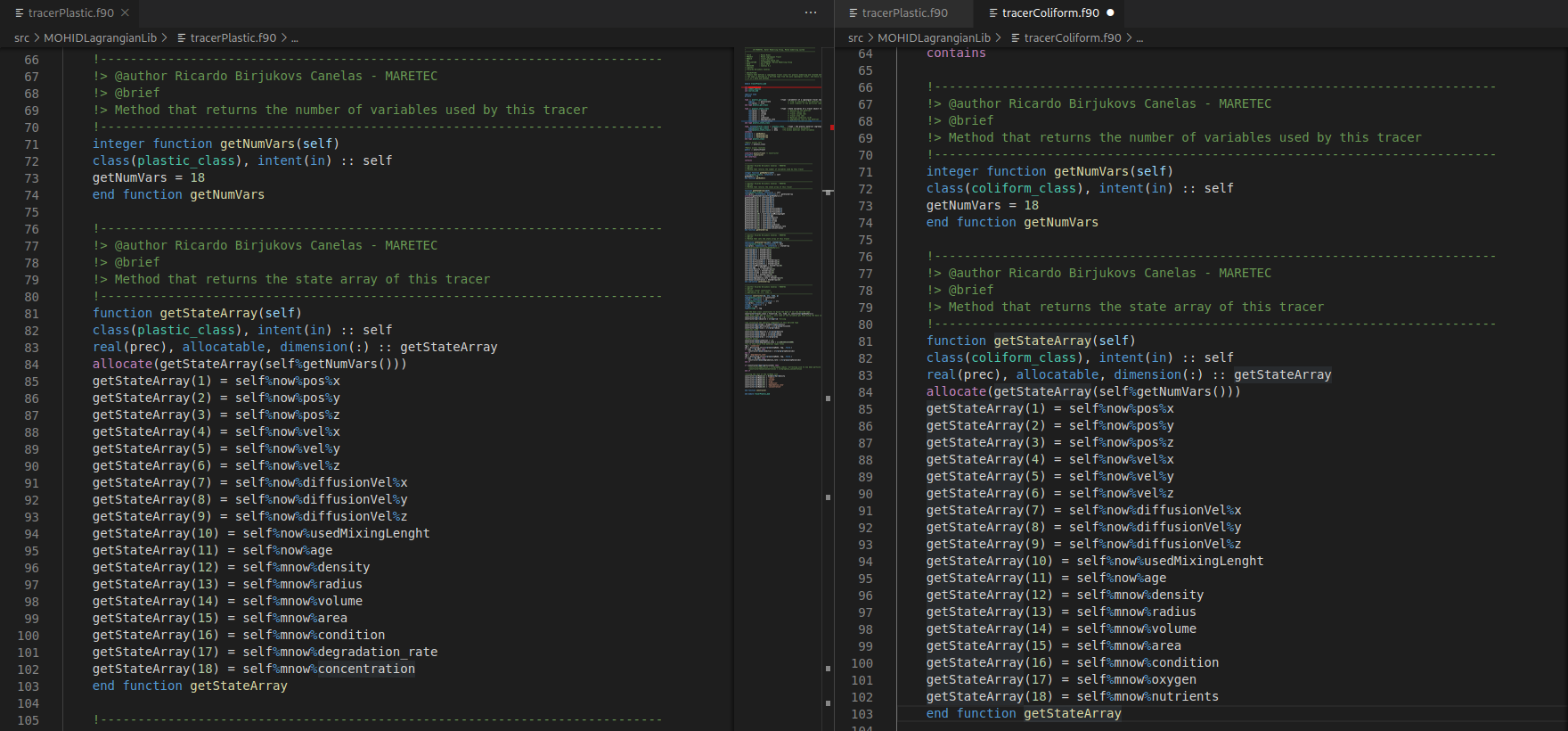
### 2.1 Introducing new properties.

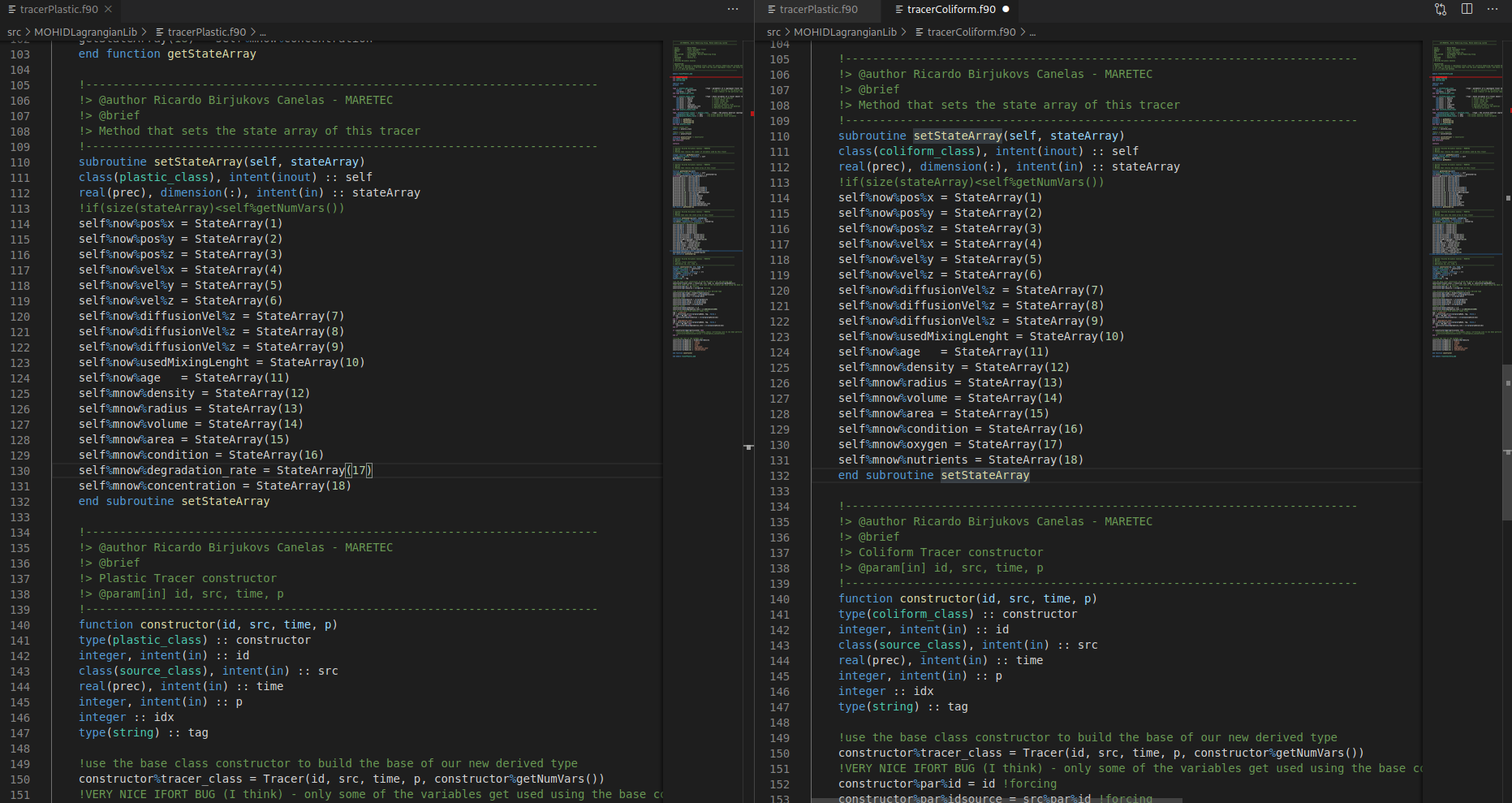
We will consider that the coliform, compared with plastic, it does not need a **degradation rat**e or **concentration**. We will delete those non useful properties and keep those which are important. Also, we can add new properties, in this example, we add (invented) two main properties such as **oxygen** and **nutrients**

### 2.2 Modifying particle properties functions

Once we define the properties, the next step is to modify the number of variables involved in the state array with the function **getNumVars. I**n this case, the number of variables remains the same, but if we change its number, we should modify according the number of variables and their index in the functions **setStateArray** and **getStateArrays**.

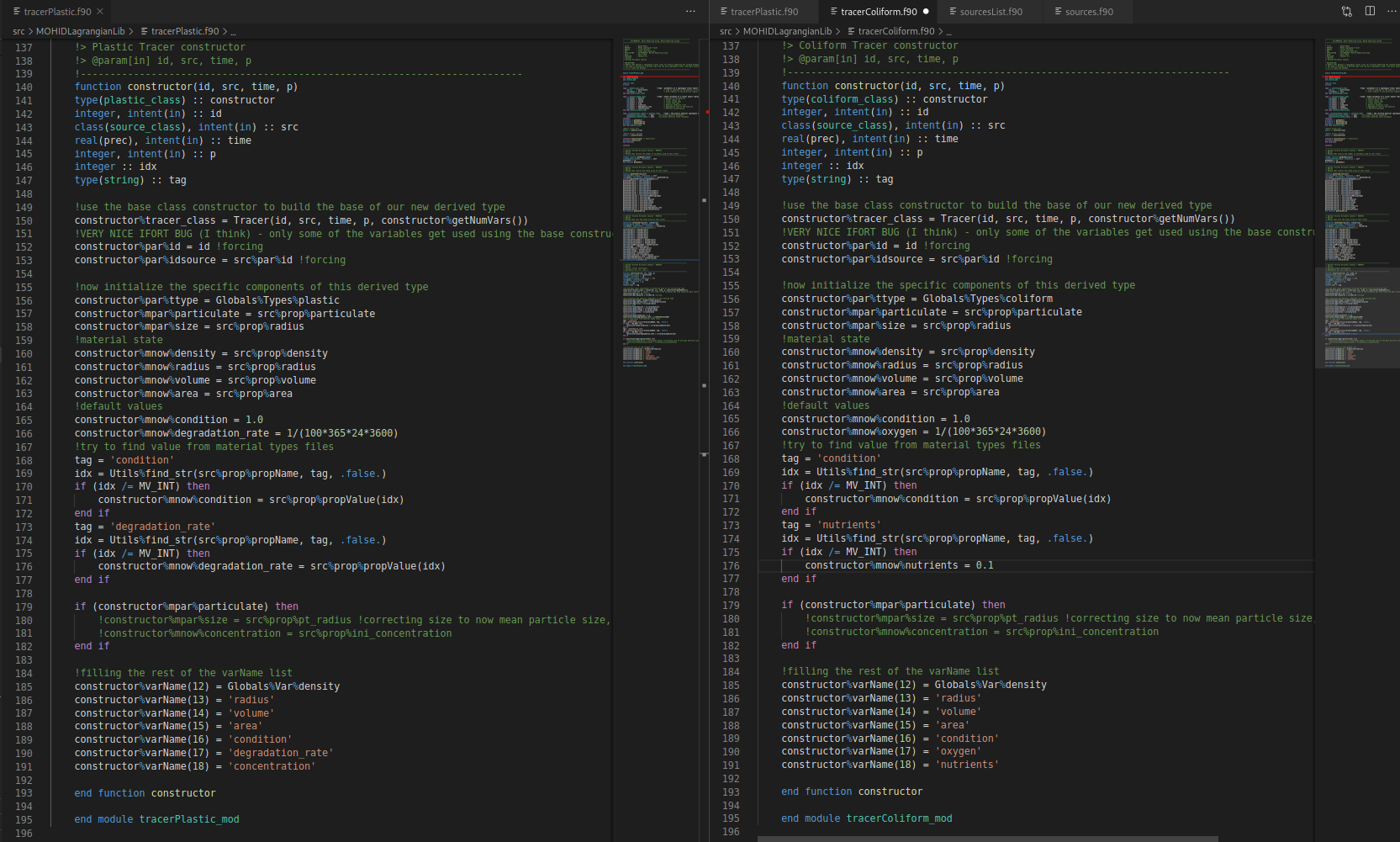
The next step is to create the setStateArray and getStateArrays. These functions stablish the correspondence between the **state array variables** and their index position inside the array to link with the coliform properties. Editing the equivalent function according to the previous variables added (oxygen and nutrients) and the previous deleted if it is the case.





Finally, the constructor class must initialize the main particle properties readed from the **Globals module,** the **source properties** or **hardcoded are also sets the variable names. Most of common names are defined by default through global variables. But those specific names should be introduced hard-coded.**

**From other side, the main values are readed from the source or are stablish by default. Keep in mind that if you want to associate a new variable with a value that starts from a source, the source must contain that attribute to be readed from the xml. In other case, just, simple add a value by the default in the code.**



Once the constructor is done, the next step is to create the functions that modifies the stateArray variables associated with the new tracer.

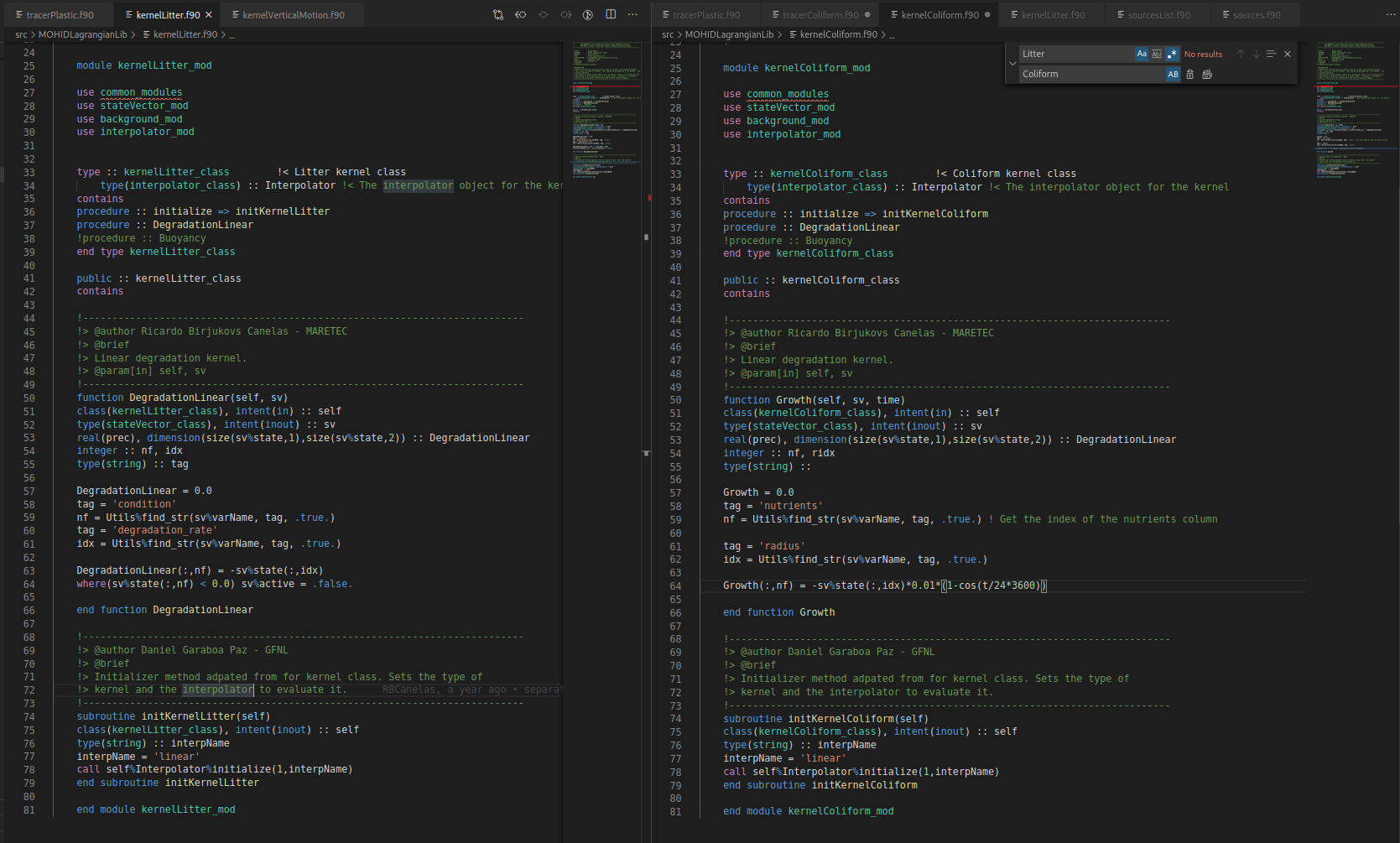
3. Creating kernels for the new particle

To that end we will create the module: module **kernelColiform.f90**. This module will contain those functions modify those specific coliform variables of the stateArray.

Let's suppose the nutrients inside our coliform depends on its size and the hour of the way, like a cosine-based function. During the day, the nutrients inside the coliforms grows up in a factor proportional to an irradiance. So the growing function with time should be something like:

*d(nutrients)/dt = R\*0.01\*(1-cos(t/24\*3600) where t is the time in seconds.*

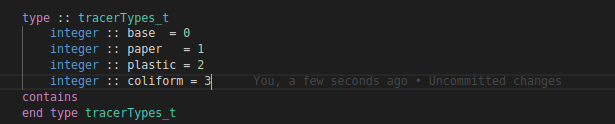
So comparing with for example. KernelLitter.f90 module, we get:



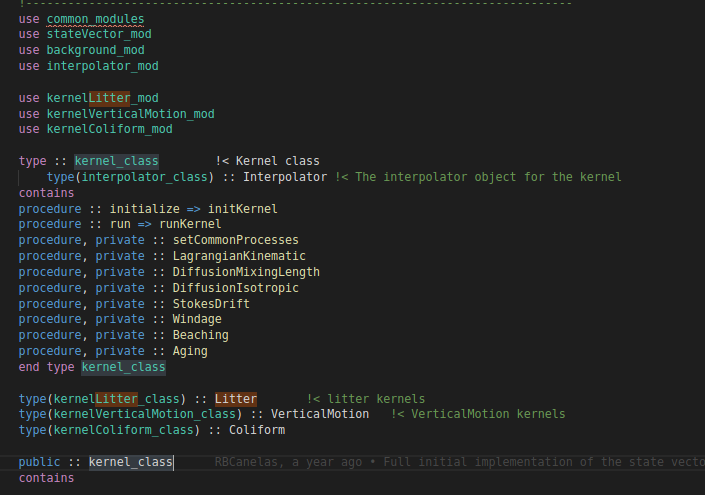
## 4. Adding this to the MOHID-Lagrangian kernel core.

Once you created a new particle type and its associated kernel or functions, we have to do two operations more.

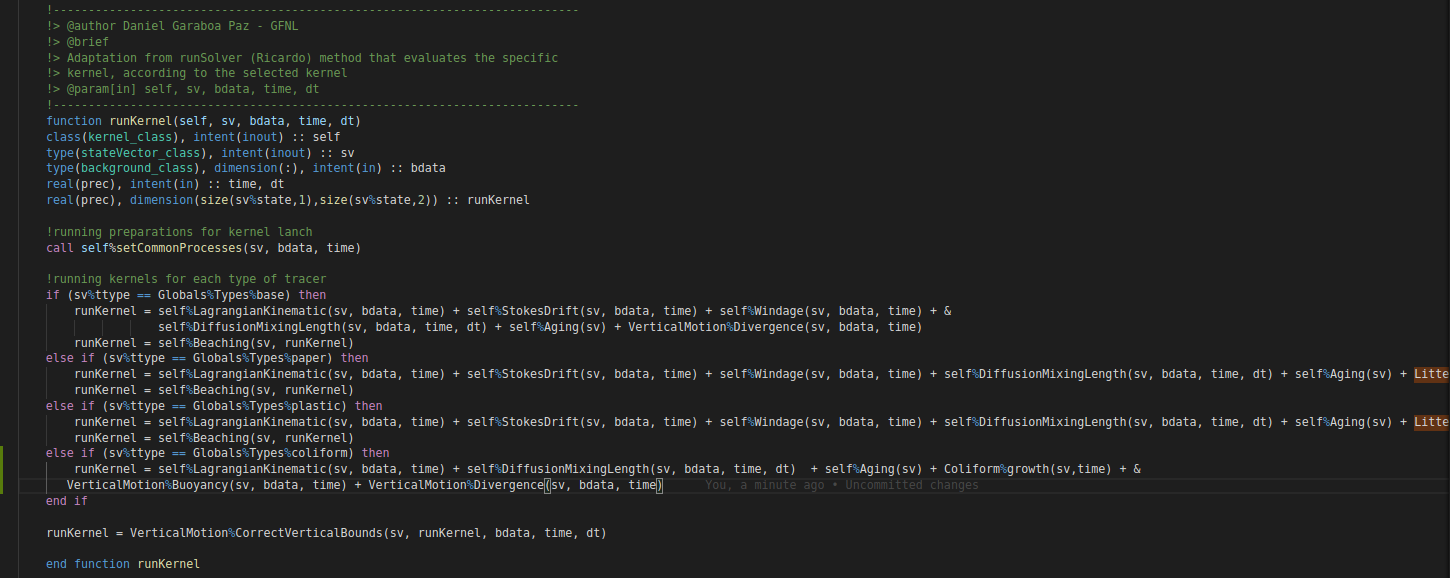
1) Add the **id** of your particle to the globals file: **simulationGlobals.f90**



2) Add it to the kernel.f90 file. First, we must add the modules created to the kernel\_module section.



Second, we add the physical process to model their behaviour. In this case, we just include the Lagrangian motion, difusion, aging (for the age of the tracer), the coliform growth and the vertical motion.



## 4. Selecting the new particle type in the xml.

Once this job is done, in the xml, in the particle type section we could select “coliform” as a new particle type and start to simulate with it.

